Figs. 8A and 8B show alternative processes to provide state-based software life cycle management using a vault. Turning now to Fig. 8A, a process for performing software management first generates the metadata (step 403), as described in Fig. 1. The metadata may include DNA information, as described in more detail in the incorporated-by-reference application. The information is then used to maintain software (step 405) before the process exits. Correspondingly, Fig. 8B shows a second software life cycle management process. Initially, the metadata information is generated and published (step 410). Next, components of the software are replicated (step 450) based on the metadata. The software is then installed (step 470). After installation, the software may be maintained (step 490).

B3

In the claims:

In the Preliminary Amendment filed April 27, 2001, claim 1 was cancelled and claims 2-22 were added.

2. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

unique instances of the one or more software components;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault; and

B4

a post controller for performing a direct, random access insertion of a software component to the vault wherein the post controller generates a unique key from the new component and optimizes storage if the unique key exists.

3. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

unique instances of the one or more software components;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault; and

a client coupled to the vault, the client having a physical software component residing on the client, the client generating a key from the physical software component.

4. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

unique instances of the one or more software components;

an access controller for performing a direct, random access retrieval of the one or more software components from the vault;

one or more secondary vaults coupled to the vault; and

a fault-tolerant rollover system for sequentially searching each vault for the presence of a target software component.

5. The computer-implemented vault of claim 4, wherein the secondary vaults are ordered based on accessibility of the vaults.

6. The computer-implemented vault of claim 4, further comprising a client for generating a key, the client applying the key to recover the target software component from the most accessible of the vaults.

7. The computer-implemented vault of claim 6, wherein the client uses a metadata description to generate the key.

8. The computer-implemented vault of claim 6, wherein the search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

9. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

means for storing unique instances of the one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault; and

a post means for performing a direct, random access insertion of a software component to the vault wherein the post means generates a unique key from the new component and optimizes storage if the unique key exists.

10. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

means for storing unique instances of the one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault; and

a client coupled to the vault, the client having a physical software component residing on the client, the client generating a key from the physical software component.

11. A computer-implemented vault for archiving software components, where only a single instance of each component that is multiply-used is stored in the vault, comprising:

BH

means for storing unique instances of the one or more software components on the vault;

access means for performing a direct, random access retrieval of the one or more software components from the vault;

one or more secondary vaults coupled to the vault; and

means for sequentially searching each vault for the presence of a target software component.

12. The computer-implemented vault of claim 10, wherein the secondary vaults are ordered based on accessibility of the vaults.

13. The computer-implemented vault of claim 10, further comprising a client for generating a key, the client having a means for applying the key to recover the target software component from the most accessible of the vaults.

14. The computer-implemented vault of claim 13, wherein the client uses a metadata description to generate the key.

15. The computer-implemented vault of claim 13, wherein the search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

16. A method for archiving software components where only a single instance of each component that is multiply-used is stored in a vault, comprising the steps of:

storing unique instances of the one or more software components in the vault; and

performing a direct, random access retrieval of the one or more software components from the vault; and

performing a direct, random access insertion of a software component to the vault wherein said step of performing an insertion generates a unique key from the new component and optimizes storage if the key exists.

B4

17. A method for archiving software components where only a single instance of each component that is multiply-used is stored in a vault, comprising the steps of:

storing unique instances of the one or more software components in the vault;

performing a direct, random access retrieval of the one or more software components from the vault; and

generating a key from a physical software component residing on a client coupled to the vault.

18. A method for archiving software components where only a single instance of each component that is multiply-used is stored in a vault, and wherein one or more secondary vaults are coupled to the vault, comprising the steps of:

storing unique instances of the one or more software components in the vault; and

performing a direct, random access retrieval of the one or more software components from the vault; and

sequentially searching each vault for the presence of a target software component.

19. The method of claim 18, wherein the secondary vaults are ordered based on accessibility of the vaults.

20. The method of claim 17, further comprising a client for generating a key, the client applying the key to recover the target software component from the most accessible of the vaults.

B4 Cond.

21. The method of claim 20, wherein the client uses a metadata description to generate the key.

22. The method of claim 20, wherein the search of a determined vault fails to locate the target software component, and wherein the client skips the determined vault and modifies the search order of the vaults in recovering the target software component.

Please enter claims 23-31.

23. (New) A method for managing one or more software components of a software application, comprising:

creating a first unique key for each of one or more respective software vaults for each software component of a software application stored on the respective software vault, each first unique key created from a metadata file associated with the respective vault and previously generated by determining run-time states of the application, each software vault remote from a first client computer on which the software application is installed;

B5

creating a second unique key from a first software component of the software application, the first software component stored on the first client computer and the second unique key containing location attributes;

comparing each of the first unique keys with the second unique key and, if the second unique key does not match any of the first unique keys, storing in one of the software vaults a copy of the first software component from the first client computer by performing a direct, random-access storage operation.

24. (New) The method of claim 23, wherein performing a storage operation comprises storing the second key along with the copy of the first software component in the first-accessed software vault.

25. (New) The method of claim 23, wherein each unique key for each respective software component is generated by:

generating metadata for the respective software component;

verifying the integrity of the respective software component and generating an integrity checksum; and

incorporating into the unique key the integrity checksum as well as information about the size, name and attributes of the respective software component.

26. (New) A method for retrieving one or more software components of a software application, comprising:

creating a unique key for a software component of a software application, the software component to be accessed from one of one or more software vaults, the unique key containing location attributes and having been created from a metadata file associated with the software component and previously generated by determining run-time states of the application, each software vault remote from a first client computer on which the software application is installed;

using the unique key to look up the software component sequentially on the one or more software vaults; and

accessing and retrieving the software component from the first software vault on which it is found.

27. (New) A method for locating one or more software components of a software application, comprising:

creating a unique key for a software component of a software application, the software component to be located in one of one or more software vaults, the unique key containing location attributes and having been created from a metadata file associated with the software

component and previously generated by determining run-time states of the application, each software vault remote from a first client computer on which the software application is installed;

determining an order of accessibility for the software vaults;

for each software vault, using the location of the software vault and the unique key, forming a uniform resource locator (URL); and

looking-up the URL in the software vaults, based on the order of accessibility, until the software component is located.

28. (New) A method for retrieving software components of a software application, comprising:

transforming a metadata description of each software component of a software application installed on a client computer into a key, each key having location attributes of the corresponding software component, the metadata description having been generated by determining run-time state of the software application; and

using the location attributes of each key, retrieving the software components from one or more software vaults accessible to the client computer through the communications network.

29. (New) The method of claim 28 comprising determining an order of accessibility of the software vaults and retrieving the software components from the most accessible software vaults.

30. (New) A method for recreating a software application, comprising:

transforming a metadata description of each software component of a software application installed on a client computer into a key, each key having location attributes of the corresponding software component, the metadata description having been generated by determining run-time state of the software application;

determining if the software components consume a large amount of file space and, upon a determination that the software components are large:

looking up a second key on the client computer;